INTERNET-DRAFT Soobok Lee
draft-ietf-idn-lsb-ace-00.txt
Expires 2001-Dec-28 2001-Jun-28

Improving ACE using code point reordering v0.9

Status of this Memo

Abstract

This document describes code point reordering to improve ACE compression algorithms. Based on frequency statistics of characater sets, this reordering is easily implemented with simple character mapping tables.

DUDE implementation of this idea shows great improvements for Hangul, Chineses, Vietnamese and European domains.

Contents

Overview
Hangul
Basic Latin
Extended Latin and Combining Diacritical Marks
Unified Han
Other character sets
Modified Encoding procedure of DUDE implementation of this idea

Modified Decoding procedure of DUDE implementation of this idea
Example strings
Security considerations
References
Author
LDUDE: Example implementation into DUDE

Overview

Pursuing shorter ACE labels is justified to save memory resources and to reduce internet traffic even for domains of average length in various application/core internet protocols.

Both 11172 Hangul syllables and 24000 or more CJK Han syllables occupy roughly half of the entire unicode space.
Their lexicographical ordering(not in frequency ordering) makes various ACE compression technique work poorly for them. Frequently used syllables are spread evenly through out those wide ranges.

Even, Latin characters code range, including 'a' - 'z' has lexicographical order that does not reflect the fact that 's','t' and 'r' are more frequently used than 'j','k' and 'h'.

Each Hangul code point integer have useful bit structure in it.
That reflects its hangul jamo(consonant/vowel) combination which provides us the hint for applying reordering
by jamo usage frequency.

Most ACE algorithms show good compression ratio when frequently used characters are re-located into narrower code areas.
Especially, to reduce DUDE XOR distance, we can make the narrow area fit in 16,256 and 4096 code-long page boundaries.

Hangul

Each code point integer of modern hangul syllables(u+ac00 ~ u+bd7f) can be decomposed to produce the following 3 jamo (hangul consonant/vowel) indices,

1) L: a leading consonant index
   in the array of modern 19 leading consonants,
2) V: a vowel index
   in the array of modern 21 vowels,
3) T: an optional trailing consonant index
   in the array of modern 28 trailing consonants.

In fact, Unicode hangul range (u+ac00 ~ u+bd7f) is a 3-dimensional array Hangul[L][V][T] with base 0xac00 and each hangul syllable has code point integer determined by L*21*28 + V*28 + T + 0xac00.

Statistically, roughly six of 28 hangul trailing consonants are frequent in korean nouns. Leading consontants and vowels show relatively even frequency distributions compared with that of trailing consonants.

A frequency mapping table f(T) for T is defined so that f(T)= FrequencyOrderOf(T) in the array of modern trailing consonants.

Hangul[L][V][T] is reordered into another 3-dimensional array Hangul[f(T)][f(V)][f(L)] with index T augmented to the highest order.

Each hangul syllable in this reorderd range has a new code point value determined by f(T)*21*19 + f(V)*19 + f(L) + 0xac00.

Average f(T) value should be sufficiently lower than f(L) and F(V), so that the lowered code point value and its induced lowered successive XOR-distances contribute to produce shorter ACE labels.

Basic Latin

Basic Latin row u+0070 ~ u+007f has 'p','r','s','t' and 'u' which are more frequently used in European nouns than '`','j','k','f' and 'g' in u+0060 ~ u+006f row which includes most frequently used 'a'~'o' .

If these two sets of 5 characters are swapped character-wise, 'p','r','s','t','u' go into the u+0060 ~ u+006f row.

Single row fits in 16 codes boundary and any character sequences from only this single row make XOR-distance or code window length shorter than 0x10 and make DUDE and other ACEs do good compression.

Extended Latin and Combining Diacritical Marks

First 6 rows from Latin Extension A(u+0100 ~ u+015f) and 6 rows from Basic Latin & Latin-1 Supplement (u+0000 ~ u+002f and u+0080 ~ u+00a0) are swapped.
First 3 rows from Combining Diacritical Marks(u+0300 ~ u+032f) and 3 rows from Latin-1 Supplement (u+00B0 ~ u+00df) are also swapped.

This makes frequently used parts of Latin Extended-A and Combining Diacritical Marks go into first 256-codes page(u+0000 ~ u+00ff). Any character sequences from this single 256-codes-long page make XOR-distance or code window length much shorter than 0x100.

This improvement benefits especially East-European and Vietnamese that use Latin Extented A and Combining Diacritical Marks.

Unified Han

CJK Unified Han syllables (u+4e00 ~ u+a48f) are ordered by their radicals.

Most frequently used 2048 Traditional/Simpilfied Chinese Han syllables are reordered into single 4096-code aligned page.

Other character sets

I have no statistical data to optimize for this code ranges, yet.

Japaneses katakana and hiragana (u+3040 ~ u+30ff) code points have lexicographical order and if we could put most frequent ones in a single row, it may greatly improve katakana-only or hiragana-only domains.

For Arabic,Hebrew,Cyrillic and Hindi, we could devise similiar optimizations, if relevant frequency statistical data are available.

Modified Encoding procedure of DUDE implementation of this idea

All ordering of nybbles and quintets is big-endian (most significant first). A nybble is 4 bits. XOR is bitwise exclusive or.

This modification is hyphen-safe.
Hyphen encoding and decoding are not affected by this modification.

```
let prev = 96
for each input integer n (in order) do begin
  if n == 45 then output hyphen minus
  else begin

    n = reorder(n)   //  ******** ADDED **********

    let diff = prev XOR n
    extract the least significant nybbles of diff, as few as are
      sufficient to hold all the nonzero bits (but at least one)
    prepend 0 to the last nybble and 1 to the rest
    output base-32 characters corresponding to the quintets
    let prev = n
  end
end
```

The encoder must either correctly handle all integer values that can be represented in the type of its input, or it must check whether the input contains values that it cannot handle and return an error if so. Under no circumstances may it produce incorrect output.

Modified Decoding procedure of DUDE implementation of this idea

```
let prev = 96
while the input string is not exhausted do begin
  if the next character is hyphen-minus then output 45
  else begin
    input characters and convert them to quintets until
      encountering a quintet beginning with 0
```

```
        fail upon encountering a non-base-32 character or end-of-input
        strip the first bit of each quintet
        concatenate the resulting nybbles to form diff
        let prev = prev XOR diff

        output restore_order(prev)  //  ******** MODIFIED **********

      end
    end
    encode the output sequence and compare it to the input string
    fail if they are not equal
```

Example strings

about 20% improvement in DUDE compression ratio is achieved in these Hangul examples.

(A) Korean Strings 1: (24 hangul syllables)

U+C138 U+ACC4 U+C758 U+BAA8 U+B4E0 U+C0AC U+B78C U+B4E4 U+C774
U+D55C U+AD6D U+C5B4 U+B97C U+C774 U+D574 U+D55C U+B2E4 U+BA74
U+C5BC U+B9C8 U+B098 U+C88B U+C744 U+AE4C

DUDE-02:
6txIy79Ny53Nz79A8wIzwwNzzuAvyIzv3AtuuIz2vBy27Jz66Iz8sI\
tusAuIyz5I23Az96Iz6zE3xAz2tD96Ry3sI (89 chars)

LDUDE :
5szf8pt3bttat3jt6iywhu3bw9qt5m37r2vmxxjxsg2mtvat3auykz\
wpxubc8xd42fw6p (70 chars)

(B) Korean Strings 2: (6 hangul syllables)

U+C138 U+ACC4 U+C758 U+BAA8 U+B4E0 U+C0AC

DUDE-02: 6txiy79ny53nz79a8wizwwn (24 chars)
LDUDE : 5szf8pt3bttat3jt6i (19 chars)

about 15%~20% improvement in DUDE compression ratio is achieved in these UniHan examples.

(C) Traditional Chinese Strings 1: (16 syllables)

u+5354 u+91c7 u+5065 u+5eb7 u+4e8b u+696d u+670d u+52d9
u+7db2 u+002d u+5354 u+91c7 u+6709 u+9650 u+516c u+53f8

DUDE-02: xvve6u3d6t4c87ctsvnuz8g8yavx7eu9ym-u88g6u3d9y6q9txj6z\
vnu3e (59 chars)

LDUDE : w84bt2sp62tc66vgu2btvi5mmtvq-t8tqt2spy6juznu5iuwg
(50 chars)

(D) Traditional Chinese Strings 2: (21 syllables)

```
    u+5317 u+4eac u+5e02 u+91ab u+85e5 u+7d93 u+6fdf u+6280
    u+8853 u+7d93 u+71df u+516c u+53f8 u+5fa1 u+91ab u+7db2
    u+7d61 u+83ef u+91ab u+7db2 u+8def

    DUDE-02: xvzht75mts4q694jtwwq92zgtuwn7xr847d9x6a6wnus5du3e6xj6\
             8sk86tj7d982qtuwe86tj9sxp ( 79 chars)

    LDUDE  : xturwzfwxiv4kywgw9jv5jt2awiuzbu3buuhuwg7ug7yjuthv3pv6\
             buwmuthwd ( 63 chars )
```

(E) Traditional Chinese Strings 3: (18 syllables)

```
    u+795e u+8fb2 u+7db2 u+990a u+8eab u+4fdd u+5065 u+7db2
    u+5065 u+5eb7 u+4e16 u+754c u+5065 u+5eb7 u+8a2d u+8a08
    u+5bb6 u+60e0

    DUDE-02: z3vq9y8n9usa8w5itz4b6tzgt95iu77hu77h87cts4bv5xkuxuj87\
             c7w3kuf7t5qv5xg ( 69 chars )

    LDUDE  : xtxkwybx2euuev9ezn694p68sb68sb66vgv4k9q69yc66vgv6gvit\
             xhw5b ( 59 chars )
```

(F) Simplified Chinese String 1 : (16 syllables)

```
     <ministry of foreign trade and economic cooperation, PRC>
     u+4e2d u+534e u+4eba u+6c11 u+5171 u+548c u+56fd u+5bf9
     u+5916 u+8d38 u+6613 u+7ecf u+6d4e u+5408 u+4f5c u+90e8

    DUDE-02: w8wpt7ydt79euu4mv7yax9puzb7seu8r7wuq85umt27ntv2bv3wgt\
             5xe795e ( 61 chars )

    LDUDE  : xsyk8zc8za4ht8ntud6saz3j5xn773p79umu67ntv2bv75e2j4m
             ( 52 chars )
```

(G) Simplified Chinese String 2 : (18 syllables)

```
    u+4e2d u+56fd u+4eba u+6c11 u+5927 u+5b66 u+4e2d u+56fd
    u+8d22 u+653f u+91d1 u+878d u+653f u+7b56 u+7814 u+7a76
    u+4e2d u+5fc3

    DUDE-02: w8wpt27at2whuu4mvxvguwbtxwmt27a757r82tp9w8qtyxn8u5ct\
             8yjvwcuycvwxmtt8q ( 73 chars )

    LDUDE  : xsyk6yk6yi4h4j5ya5yn6yk7twc76vjwcywhysfu7iu8rxktznwj
             (53 chars )
```

LDUDE show better compression ratios for other scripts,too.

(H) Vietnamese: (38 syllables using diacritical marks)

```
    Ta<dotbelow>isaoho<dotbelow>kh<ocirc>ngth<ecirc><hookabove>chi\
    <hookabove>no<acute>iti<ecirc><acute>ngVi<ecirc><dotbelow>t
    U+0054 u+0061 u+0323 u+0069 u+0073 u+0061 u+006F u+0068 u+006F
    u+0323 u+006B u+0068 u+00F4 u+006E u+0067 u+0074 u+0068 u+00EA
    u+0309 u+0063 u+0068 u+0069 u+0309 u+006E u+006F u+0301 u+0069
```

u+0074 u+0069 u+00EA u+0301 u+006E u+0067 U+0056 u+0069 u+00EA
u+0323 u+0074

DUDE-02:
vEvfvwcvwktktcqhhvwnvwid3n3kjtdtn2cv8dvykmbvyavyhbvyqv\
yitptp2dv8mvyrjvBvr2dv6jvxh (82 chars)
LDUDE :
uGuh5c5kckqhh5n4atm3n3ktmtdq2cxd7kmb7a7hb7q7irr2dxm7rt\
muDvr2dvj5f (66 chars , 16 chars(19%) shorter)

(I) Spanish: (using basic Latin & Latin Supplement)
Porqu<eacute>nopuedensimplementehablarenEspa<ntilde>ol
U+0050 u+006F u+0072 u+0071 u+0075 u+00E9 u+006E u+006F u+0070
u+0075 u+0065 u+0064 u+0065 u+006E u+0073 u+0069 u+006D u+0070
u+006C u+0065 u+006D u+0065 u+006E u+0074 u+0065 u+0068 u+0061
u+0062 u+006C u+0061 u+0072 u+0065 u+006E U+0045 u+0073 u+0070
u+0061 u+00F1 u+006F u+006C

DUDE-02:
vAvrtpde3n2hbtrftabbmtptketptnjiimtktbpjdqptdthmuMvgdt\
b3a3qd (61 chars)
LDUDE : uAurftmtg2q2hbrhcbbmfcepnjiimidpjdqpmrmuMuqmb3a3qd
(51 chars, 10 chars (16%) shorter)

(J) Czech: (using Latin Extended A)
Pro<ccaron>prost<ecaron>nemluv<iacute><ccaron>esky
U+0050 u+0072 u+006F u+010D u+0070 u+0072 u+006F u+0073 u+0074
u+011B u+006E u+0065 u+006D u+006C u+0075 u+0076 u+00ED u+010D
u+0065 u+0073 u+006B u+0079

DUDE-02: vAuctptyctzpctptnhtyrtzfmibtjd3mt8atyitgtitc (45
chars)

LDUDE : uAukfycypkfepzpzfmibmtb3m8ayiqtik(34 chars,24%
shorter)

Security considerations

ACE-encoded reordered code points are restored in reverse ACE translation with no problem, and this improvement do not introduce any new security problems into ACE.

References

[DUDE02] Mark Welter, Brian Spolarich, Adam Costello, "DUDE: Differential Unicode Domain Encoding", 2001-May-31, draft-ietf-idn-dude-02.

[AMCACEW] Adam Costello, "AMC-ACE-W version 0.1.0", 2001-May-31, draft-ietf-idn-amc-ace-w-00, latest version at http://www.cs.berkeley.edu/~amc/charset/amc-ace-w.

[UNICODE] The Unicode Consortium, "The Unicode Standard",
http://www.unicode.org/unicode/standard/standard.html.

[IDNA] Patrik Falstrom, Paul Hoffman, "Internationalizing Host
Names In Applications (IDNA)", draft-ietf-idn-idna-01

[NAMEPREP] Paul Hoffman, Marc Blanchet, "Preparation of
Internationalized Host Names", Feb 2001,
draft-ietf-idn-nameprep-03

Author

Soobok Lee <lsb@postel.co.kr>
Postel Services, Inc.
http://www.postel.co.kr
Tel: +82-11-9774-2737

Example implementation

This idea is applicable to any ACEs.
LDUDE (in this example implementation) is merely a name for
DUDE implementation of this idea.

Embedded hangul jamo and Latin frequency tables are subject
to change with further studies in the next revision of this draft.

In Unix, save this example source code into ldude.c

```
% cc -o ldude ldude.c
% ./ldude -e < input_file > output_file
% ./ldude -d < output_file
```

A input file should contains u+????-form code points
delimited with spaces or newlines.

```
/*****************************************/
/* ldude.c 1.0  (2001-Jun-28)            */
/* Soobok  Lee  <lsb@postel.co.kr>       */
/* Adam M. Costello <amc@cs.berkeley.edu> */
/*****************************************/

/* This is ANSI C code (C89) implementing */
/* DUDE (draft-ietf-idn-ldude-01).        */


/*********************************************************/
/* Public interface (would normally go in its own .h file): */

#include <stdio.h>
#include <limits.h>
```

```
enum dude_status {
  dude_success,
  dude_bad_input,
  dude_big_output  /* Output would exceed the space provided. */
};

enum case_sensitivity { case_sensitive, case_insensitive };

#if UINT_MAX >= 0x1FFFFF
typedef unsigned int u_code_point;
#else
typedef unsigned long u_code_point;
#endif

enum dude_status dude_encode(
  unsigned int input_length,
  const u_code_point input[],
  const unsigned char uppercase_flags[],
  unsigned int *output_size,
  char output[] );

    /* dude_encode() converts Unicode to DUDE (without any
*/
    /* signature).  The input must be represented as an array
*/
    /* of Unicode code points (not code units; surrogate pairs
*/
    /* are not allowed), and the output will be represented as
*/
    /* null-terminated ASCII.  The input_length is the number of code
*/
    /* points in the input.  The output_size is an in/out argument:
*/
    /* the caller must pass in the maximum number of characters
*/
    /* that may be output (including the terminating null), and on
*/
    /* successful return it will contain the number of characters
*/
    /* actually output (including the terminating null, so it will be
*/
    /* one more than strlen() would return, which is why it is called
*/
    /* output_size rather than output_length).  The uppercase_flags
*/
    /* array must hold input_length boolean values, where nonzero
*/
    /* means the corresponding Unicode character should be forced
*/
    /* to uppercase after being decoded, and zero means it is
*/
    /* caseless or should be forced to lowercase.  Alternatively,
*/
    /* uppercase_flags may be a null pointer, which is equivalent
*/
```

```
    /* to all zeros.  The encoder always outputs lowercase base-32
*/
    /* characters except when nonzero values of uppercase_flags
*/
    /* require otherwise.  The return value may be any of the
*/
    /* dude_status values defined above; if not dude_success, then
*/
    /* output_size and output may contain garbage.  On success, the
*/
    /* encoder will never need to write an output_size greater than
*/
    /* input_length*k+1 if all the input code points are less than 1
*/
    /* << (4*k), because of how the encoding is defined.
*/

enum dude_status dude_decode(
  enum case_sensitivity case_sensitivity,
  char scratch_space[],
  const char input[],
  unsigned int *output_length,
  u_code_point output[],
  unsigned char uppercase_flags[] );

    /* dude_decode() converts DUDE (without any signature) to
*/
    /* Unicode.  The input must be represented as null-terminated
*/
    /* ASCII, and the output will be represented as an array of
*/
    /* Unicode code points.  The case_sensitivity argument influences
*/
    /* the check on the well-formedness of the input string; it
*/
    /* must be case_sensitive if case-sensitive comparisons are
*/
    /* allowed on encoded strings, case_insensitive otherwise.
*/
    /* The scratch_space must point to space at least as large
*/
    /* as the input, which will get overwritten (this allows the
*/
    /* decoder to avoid calling malloc()).  The output_length is
*/
    /* an in/out argument: the caller must pass in the maximum
*/
    /* number of code points that may be output, and on successful
*/
    /* return it will contain the actual number of code points
*/
    /* output.  The uppercase_flags array must have room for at
*/
    /* least output_length values, or it may be a null pointer if
*/
    /* the case information is not needed.  A nonzero flag indicates
```

```
*/
    /* that the corresponding Unicode character should be forced to
*/
    /* uppercase by the caller, while zero means it is caseless or
*/
    /* should be forced to lowercase.  The return value may be any
*/
    /* of the dude_status values defined above; if not dude_success,
*/
    /* then output_length, output, and uppercase_flags may contain
*/
    /* garbage.  On success, the decoder will never need to write
*/
    /* an output_length greater than the length of the input (not
*/
    /* counting the null terminator), because of how the encoding is
*/
    /* defined.
*/


/*********************************************************/
/* Implementation (would normally go in its own .c file): */

#include <string.h>

/* Character utilities: */

/* base32[q] is the lowercase base-32 character representing  */
/* the number q from the range 0 to 31.  Note that we cannot  */
/* use string literals for ASCII characters because an ANSI C */
/* compiler does not necessarily use ASCII.                   */

static const char base32[] = {
  97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,     /* a-k */
  109, 110,                                               /* m-n */
  112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122,  /* p-z */
  50, 51, 52, 53, 54, 55, 56, 57                          /* 2-9 */
};

/* base32_decode(c) returns the value of a base-32 character, in the */
/* range 0 to 31, or the constant base32_invalid if c is not a valid */
/* base-32 character.                                                */

enum { base32_invalid = 32 };

static unsigned int base32_decode(char c)
{
  if (c < 50) return base32_invalid;
  if (c <= 57) return c - 26;
  if (c < 97) c += 32;
  if (c < 97 || c == 108 || c == 111 || c > 122) return base32_invalid;
  return c - 97 - (c > 108) - (c > 111);
}

/* unequal(case_sensitivity,s1,s2) returns 0 if the strings s1 and s2
```

```
*/
/* are equal, 1 otherwise.  If case_sensitivity is case_insensitive,
*/
/* then ASCII A-Z are considered equal to a-z respectively.
*/

static int unequal( enum case_sensitivity case_sensitivity,
                    const char s1[], const char s2[]        )
{
  char c1, c2;

  if (case_sensitivity != case_insensitive) return strcmp(s1,s2) != 0;

  for (;;) {
    c1 = *s1;
    c2 = *s2;
    if (c1 >= 65 && c1 <= 90) c1 += 32;
    if (c2 >= 65 && c2 <= 90) c2 += 32;
    if (c1 != c2) return 1;
    if (c1 == 0) return 0;
    ++s1, ++s2;
  }
}



/* LANGUAGE-SPECIFIC IMPROVEMENTS TO DUDE  BASED ON CODE REORDERING */

/* Common Constants for Hangul */
static u_code_point
        SBase = 0xAC00, LBase = 0x1100, VBase = 0x1161, TBase = 0x11A7,
        LCount = 19, VCount = 21, TCount = 28, NCount, SCount;
static u_code_point NCount = 588;   // VCount * TCount
static u_code_point SCount = 11172; // LCount * NCount
static u_code_point MCount = 399; // LCount * VCount

int JAMO_L_FREQ[19][2] = {
        {1,11}, // "G"    0
        {14,0}, // "GG"
        {9,9}, // "N"
        {5,12}, // "D"
        {15,7}, // "DD"
        {13,3}, // "R"    5
        {7,18}, // "M"
        {4,6 }, // "B"
        {16,14}, // "BB"
        {2, 2}, // "S"
        {17,17}, // "SS" 10
        {0,16}, // "O"
        {3,15}, // "J"
        {18,5 }, // "JJ"
        {8,1}, // "C"
        {12,4}, // "K"   15
        {11,8}, // "T"
        {10,10}, // "P"
        {6 ,13}  // "H"
```

```
   };

int JAMO_V_FREQ[21][2] = {
       {2,20}, // "A"    0
       {7,5}, // "AE"
       {15,0}, // "YA"
       {20,13}, // "YAE"
       {5,18}, // "EO"
       {1, 4}, // "E"     5
       {9,8}, // "YEO"
       {13,1},// "YE"
       {6,13}, // "O"
       {10, 6}, // "WA"
       {18,9}, // "WAE"  10
       {16,17}, // "OE"
       {17,14}, // "YO"
       {8, 7}, // "U"
       {12,16}, // "WEO"
       {19, 2}, // "WE"   15
       {14,11}, // "WI"
       {11,12}, // "YU"
       {4,10}, // "EU"
       {19,19}, // "YI"
       {0,3}  // "I"     20
   };

 int JAMO_T_FREQ[28][2] = {
       {0,0}, // ""      0
       {5,4}, // "G"
       {15,21}, // "GG"
       {16,8}, // "GS"
       {1,16}, // "N"
       {27,1 }, // "NJ"   5
       {17,17}, // "NH"
       {10,19}, // "D"
       {3,24}, // "L"
       {18,27}, // "LG"
       {19, 7}, // "LM"   10
       {20,22}, // "LB"
       {21,25}, // "LS"
       {22,26}, // "LT"
       {23,27}, // "LP"
       {24,2}, // "LH"   15
       {4,3}, // "M"
       {6,6}, // "B"
       {25,9}, // "BS"
       {7,10}, // "S"
       {26,11}, // "SS"   20
       {2,12}, // "NG"
       {11,13}, // "J"
       {10,14}, // "C"
       {8 ,15}, // "K"
       {12,18}, // "T"    25
       {13,20}, // "P"
       {9 ,5}  // "H"    27
   };
```

```
/* Hangul Decomposition */


int isHANGUL(u_code_point s) {
        int SIndex = s - SBase;
        if (SIndex < 0 || SIndex >= SCount) {
            return 0;
        }
        return 1;
};
int isUNIHAN(u_code_point s) {
        if (s >= 0x4E00 && s <= 0x9FAF) {
            return 1;
        }
        return 0;
};
int isLatins(u_code_point s) {
        if (s <  0x370) {
            return 1;
        }
        return 0;
};

u_code_point reorder_hangul(u_code_point s) {
        u_code_point SIndex = s - SBase;
        int zL = JAMO_L_FREQ[SIndex / NCount][0];
        int zV = JAMO_V_FREQ[(SIndex % NCount) / TCount][0];
        int zT = JAMO_T_FREQ[SIndex % TCount][0];
        int idx= (zT*MCount +  zV*LCount +  zL);
        if(idx < SCount-0x400) {
           idx += 0x400;
        } else {
           idx  = idx - (SCount -0x400);
        };
        return  idx + SBase;
}

u_code_point restore_order_hangul(u_code_point z) {
        int T,V,L;
        u_code_point zIndex = z-SBase;
        if(zIndex < 0x400) {
           zIndex += SCount - 0x400;
        } else {
           zIndex  = zIndex - 0x400;
        };
        T = JAMO_T_FREQ[(zIndex / MCount)][1];
        V = JAMO_V_FREQ[(zIndex % MCount) / LCount][1];
        L = JAMO_L_FREQ[(zIndex % LCount)][1];
        return (L*NCount + V*TCount + T) + SBase;
}

#define U8(A,B,C,D,E,F,G,H) U(A);U(B);U(C);U(D);U(E);U(F);U(G);U(H);
#define U(A) if(j==A) return(i); if(j==i) return(A); i++;

u_code_point reorder_unihan(u_code_point s) {
```

```
        register u_code_point i=0x5000;
        register u_code_point j=s;

// TC : most frequent 2048

        U8(0x7684,0x662f,0x4e0d,0x6211,0x4e00,0x6709,0x5927,0x5728);
        U8(0x4eba,0x4e86,0x4e2d,0x5230,0x8cc7,0x8981,0x4ee5,0x53ef);
        U8(0x9019,0x500b,0x4f60,0x6703,0x597d,0x70ba,0x4e0a,0x4f86);
        U8(0x5b78,0x5c31,0x4ea4,0x4e5f,0x7528,0x80fd,0x5982,0x6642);
        U8(0x6587,0x8aaa,0x6c92,0x4ed6,0x770b,0x90a3,0x554f,0x751f);
        U8(0x63d0,0x4e0b,0x904e,0x8acb,0x5011,0x5929,0x6240,0x591a);
        U8(0x9ebc,0x5c0f,0x4e4b,0x60f3,0x5f97,0x5de5,0x51fa,0x9084);
        U8(0x96fb,0x5c0d,0x90fd,0x6a5f,0x81ea,0x800c,0x5b50,0x5f8c);
        U8(0x8a0a,0x5bb6,0x7ad9,0x5fc3,0x53ea,0x53bb,0x77e5,0x570b);
        U8(0x5f88,0x53f0,0x6210,0x4fe1,0x540c,0x4f55,0x7ae0,0x9053);
        U8(0x767c,0x5730,0x6cd5,0x7121,0x7136,0x4f46,0x7576,0x65bc);
        U8(0x55ce,0x672c,0x5e74,0x73fe,0x524d,0x6700,0x771f,0x65b0);
        U8(0x548c,0x56e0,0x679c,0x610f,0x5b9a,0x9ede,0x60c5,0x5176);
        U8(0x984c,0x4e8b,0x79d1,0x65b9,0x4e9b,0x6e05,0x4e09,0x6a23);
        U8(0x6b64,0x5427,0x4f4d,0x4f5c,0x7406,0x884c,0x8005,0x7d93);
        U8(0x540d,0x4ec0,0x8b1d,0x65e5,0x6b63,0x958b,0x8a71,0x8207);
        U8(0x5be6,0x611b,0x518d,0x83ef,0x4e8c,0x57ce,0x52d5,0x6bd4);
        U8(0x9762,0x9ad8,0x53c8,0x6216,0x529b,0x61c9,0x5973,0x7a2e);
        U8(0x6559,0x8eca,0x5206,0x50cf,0x7cfb,0x9577,0x624b,0x6b21);
        U8(0x5df2,0x660e,0x6253,0x592a,0x8def,0x8d77,0x5df1,0x76f8);
        U8(0x4e3b,0x95dc,0x5341,0x9593,0x9cf3,0x5916,0x5462,0x89ba);
        U8(0x4f7f,0x8a72,0x53cb,0x624d,0x9032,0x51f0,0x5979,0x6c11);
        U8(0x8457,0x5404,0x5168,0x5c07,0x5c11,0x5169,0x52a0,0x56de);
        U8(0x611f,0x5f0f,0x7b2c,0x7403,0x6027,0x8001,0x7a0b,0x628a);
        U8(0x88ab,0x516c,0x8ad6,0x53ca,0x9f8d,0x6821,0x5225,0x9ad4);
        U8(0x91cd,0x7d66,0x807d,0x6c34,0x505a,0x5e38,0x60a8,0x898b);
        U8(0x88e1,0x6771,0x98a8,0x89e3,0x7063,0x6708,0x7b49,0x5566);
        U8(0x90e8,0x539f,0x7f8e,0x5148,0x97f3,0x901a,0x7ba1,0x7db2);
        U8(0x5340,0x671f,0x932f,0x5426,0x6a02,0x5165,0x627e,0x66f8);
        U8(0x8b93,0x56db,0x554a,0x7531,0x9078,0x8f03,0x6578,0x8868);
        U8(0x5167,0x5834,0x5b83,0x5f9e,0x5feb,0x6b61,0x81f3,0x7acb);
        U8(0x76ee,0x793e,0x5408,0x671b,0x600e,0x8a8d,0x544a,0x66f4);
        U8(0x5e7e,0x8003,0x5ea6,0x96e3,0x7248,0x982d,0x559c,0x8a31);
        U8(0x5149,0x4eca,0x8cb7,0x7b97,0x5f1f,0x82e5,0x7d71,0x8eab);
        U8(0x8a18,0x4ee3,0x865f,0x8655,0x5b8c,0x63a5,0x8a08,0x8a00);
        U8(0x5b57,0x5e2b,0x4e26,0x653f,0x73a9,0x5f35,0x7537,0x8ab0);
        U8(0x5c71,0x6bcf,0x7d50,0x4e14,0x661f,0x975e,0x5efa,0x6539);
        U8(0x9023,0x653e,0x54c8,0x6d3b,0x7814,0x76f4,0x8a2d,0x9673);
        U8(0x5831,0x8f49,0x9ee8,0x6307,0x4e94,0x8b8a,0x6c23,0x897f);
        U8(0x8a66,0x5e0c,0x795e,0x53d6,0x5316,0x7269,0x738b,0x4efb);
        U8(0x6797,0x55ae,0x4e16,0x53d7,0x8fd1,0x7fa9,0x6b7b,0x4fbf);
        U8(0x53cd,0x58eb,0x6230,0x7a7a,0x968a,0x8ddf,0x537b,0x5317);
        U8(0x5fc5,0x696d,0x529f,0x5beb,0x5f71,0x8072,0x5e73,0x81fa);
        U8(0x54e1,0x91d1,0x8a0e,0x8272,0x5247,0x5bb9,0x6a94,0x7247);
        U8(0x5411,0x59b3,0x5e02,0x5229,0x8208,0x767d,0x5f37,0x5b89);
        U8(0x592e,0x7279,0x8b70,0x8fa6,0x50f9,0x7e3d,0x50b3,0x601d);
        U8(0x82b1,0x5143,0x53eb,0x4fdd,0x4efd,0x6c42,0x7a76,0x5475);
        U8(0x4ef6,0x672a,0x6c7a,0x7d44,0x842c,0x7af9,0x7d1a,0x6301);
        U8(0x7b11,0x6295,0x54ea,0x5ba4,0x66fe,0x8d70,0x5594,0x6a19);
        U8(0x6d41,0x652f,0x7368,0x8c93,0x5361,0x9700,0x5144,0x9580);
        U8(0x5171,0x8a9e,0x6d77,0x53e3,0x963f,0x7dda,0x99ac,0x9ec3);
```

```
U8(0x53c3,0x822c,0x547d,0x8996,0x89c0,0x806f,0x8166,0x670b);
U8(0x683c,0x5152,0x516b,0x4fee,0x6599,0x9322,0x5931,0x5403);
U8(0x4f4f,0x5373,0x53e6,0x9304,0x5c08,0x8c61,0x63db,0x57fa);
U8(0x677f,0x62ff,0x9060,0x901f,0x5f62,0x5b69,0x5099,0x6b4c);
U8(0x5e6b,0x78ba,0x5019,0x9664,0x754c,0x88dd,0x985e,0x8b1b);
U8(0x5668,0x5357,0x6848,0x756b,0x82f1,0x8a34,0x5e36,0x5dee);
U8(0x4e4e,0x91cf,0x4e45,0x6389,0x4f3c,0x6574,0x5f15,0x73ed);
U8(0x8ff7,0x5716,0x5236,0x8cbb,0x8cfd,0x5947,0x8b58,0x578b);
U8(0x8d85,0x908a,0x8036,0x54c1,0x820d,0x96d6,0x59cb,0x904b);
U8(0x674e,0x52d9,0x6b0a,0x9a57,0x6545,0x516d,0x8b80,0x602a);
U8(0x98db,0x6eff,0x670d,0x5922,0x6536,0x773c,0x9020,0x5ff5);
U8(0x7559,0x8ab2,0x8ecd,0x7834,0x7cbe,0x534a,0x7d04,0x9858);
U8(0x4ee4,0x5e95,0x7b54,0x6f14,0x9054,0x96c4,0x6df1,0x7968);
U8(0x65e9,0x9662,0x5920,0x66f2,0x5047,0x8ac7,0x8853,0x68d2);
U8(0x8ce3,0x9ed1,0x767e,0x52dd,0x63a8,0x5b58,0x706b,0x6e96);
U8(0x793a,0x5f80,0x789f,0x6613,0x6cc1,0x665a,0x96e2,0x6cbb);
U8(0x5c0e,0x4e03,0x6bb5,0x5718,0x8abf,0x8b49,0x5217,0x50b7);
U8(0x6c38,0x525b,0x6392,0x54e5,0x5fb7,0x4e5d,0x751a,0x6bba);
U8(0x7167,0x8edf,0x5305,0x6015,0x689d,0x591c,0x5546,0x6982);
U8(0x6839,0x4f9b,0x7d55,0x5343,0x5ba2,0x5207,0x96c6,0x7a31);
U8(0x64da,0x843d,0x8d8a,0x7adf,0x76e1,0x5f85,0x805e,0x5712);
U8(0x5fd8,0x503c,0x7522,0x6d88,0x96d9,0x7d05,0x5ea7,0x5c55);
U8(0x80b2,0x8dd1,0x9644,0x561b,0x57f7,0x5531,0x6280,0x67d0);
U8(0x786c,0x65af,0x96f2,0x904a,0x606f,0x52a9,0x9808,0x82e6);
U8(0x4ecb,0x6548,0x9996,0x8cea,0x4f8b,0x5509,0x8077,0x5fa9);
U8(0x8f38,0x7bc0,0x898f,0x6ce8,0x7562,0x67e5,0x71b1,0x6cb9);
U8(0x9928,0x614b,0x505c,0x798f,0x6551,0x5012,0x89aa,0x5bb3);
U8(0x4e82,0x53e4,0x6b65,0x5bf6,0x64ca,0x8209,0x7d42,0x55ef);
U8(0x5370,0x9650,0x4f9d,0x65b7,0x8f15,0x74b0,0x7c21,0x8da3);
U8(0x5fd7,0x97ff,0x96a8,0x7df4,0x7e8c,0x9b5a,0x7bc7,0x53f8);
U8(0x5c40,0x9001,0x6975,0x89d2,0x7701,0x6e90,0x967d,0x5e79);
U8(0x7fd2,0x7f85,0x6b66,0x514d,0x7591,0x62c9,0x514b,0x4ecd);
U8(0x6a13,0x4f5b,0x8db3,0x4f4e,0x5ee3,0x7169,0x9ce5,0x986f);
U8(0x78bc,0x571f,0x7387,0x8056,0x58de,0x521d,0x5177,0x9810);
U8(0x5440,0x773e,0x8cac,0x722d,0x5175,0x667a,0x8aa4,0x5883);
U8(0x9752,0x9806,0x91ce,0x695a,0x8cb4,0x8ca0,0x58d3,0x53f2);
U8(0x9069,0x4fc2,0x6e2c,0x61f7,0x8fce,0x914d,0x9b54,0x6162);
U8(0x54c7,0x61c2,0x55da,0x4ea6,0x5473,0x8a55,0x821e,0x7d30);
U8(0x91ab,0x5e1d,0x5c6c,0x53e5,0x6200,0x6557,0x5b9c,0x694a);
U8(0x7532,0x8ffd,0x704c,0x6625,0x5de6,0x6562,0x9748,0x72c2);
U8(0x969b,0x7fa4,0x65cf,0x6728,0x9a0e,0x91cc,0x9805,0x6232);
U8(0x9047,0x72d7,0x4f73,0x535a,0x53f3,0x75db,0x71df,0x59b9);
U8(0x5eb7,0x5584,0x5fb5,0x6b77,0x5b98,0x723e,0x6309,0x7de8);
U8(0x75c5,0x8b77,0x88dc,0x64c7,0x6293,0x77f3,0x6b72,0x96bb);
U8(0x9818,0x5c0b,0x6eab,0x990a,0x6b62,0x5b88,0x541b,0x8840);
U8(0x7530,0x96e8,0x5c45,0x8b02,0x7570,0x512a,0x8df3,0x62dc);
U8(0x721b,0x5c01,0x60e1,0x826f,0x6a21,0x72c0,0x6d6a,0x804a);
U8(0x589e,0x6838,0x6fc0,0x7dad,0x9678,0x5433,0x725b,0x5fd9);
U8(0x8a5e,0x5287,0x5bbf,0x6025,0x5565,0x62b1,0x975c,0x653b);
U8(0x4e9e,0x6c5f,0x81f4,0x9663,0x56b4,0x5b97,0x8b66,0x58d8);
U8(0x592b,0x5bc6,0x7761,0x5348,0x5e97,0x52e2,0x60b2,0x862d);
U8(0x5e55,0x7de3,0x9031,0x5ee0,0x7c3d,0x5750,0x9999,0x723d);
U8(0x63a7,0x5fae,0x767b,0x7ffb,0x666e,0x883b,0x51b7,0x5a01);
U8(0x6bd2,0x4fca,0x7d61,0x8f2f,0x6bcd,0x5275,0x5802,0x8d99);
U8(0x5957,0x820a,0x96dc,0x5468,0x8ff0,0x6050,0x5e78,0x4eae);
U8(0x9e97,0x5df4,0x79ae,0x9152,0x4ec1,0x9910,0x724c,0x7a81);
```

```
U8(0x8173,0x528d,0x62db,0x5409,0x7236,0x4ed4,0x5178,0x641e);
U8(0x623f,0x7d20,0x9632,0x6388,0x5145,0x8349,0x66b4,0x616e);
U8(0x7d39,0x80cc,0x5289,0x59d4,0x5e9c,0x666f,0x61b6,0x5c24);
U8(0x8af8,0x7f3a,0x63f4,0x6f2b,0x7434,0x7f75,0x7d14,0x5c1a);
U8(0x85dd,0x60dc,0x7f6e,0x76ca,0x59d0,0x8aa0,0x7e7c,0x6e56);
U8(0x6b32,0x9ebb,0x9760,0x8089,0x677e,0x523b,0x7d00,0x9000);
U8(0x65e2,0x542b,0x5224,0x91cb,0x76ae,0x6ce2,0x627f,0x5c04);
U8(0x5806,0x83ab,0x88fd,0x9375,0x8d95,0x65c1,0x7b46,0x6241);
U8(0x8a3b,0x594f,0x6a39,0x5f8b,0x9435,0x69ae,0x6628,0x6bdb);
U8(0x5f69,0x6b78,0x864e,0x7f6a,0x7686,0x8449,0x552e,0x5f48);
U8(0x885b,0x65bd,0x9298,0x5200,0x584a,0x6f22,0x6b23,0x5e03);
U8(0x8cde,0x8f09,0x96aa,0x64ad,0x5347,0x9418,0x5bc4,0x5f04);
U8(0x4ed8,0x69cb,0x56c9,0x78c1,0x87a2,0x5049,0x85a6,0x6d0b);
U8(0x563f,0x555f,0x6885,0x7b56,0x563b,0x71c8,0x9b3c,0x6aa2);
U8(0x5ba3,0x54e6,0x5abd,0x5747,0x6d3e,0x8c6c,0x6fdf,0x67b6);
U8(0x4eab,0x5446,0x8a13,0x85cd,0x5283,0x64d4,0x52aa,0x90ed);
U8(0x6b49,0x7d19,0x8cbc,0x6697,0x547c,0x7f77,0x5de7,0x6167);
U8(0x7a7f,0x8a73,0x96f7,0x5354,0x7763,0x9867,0x81c9,0x9022);
U8(0x5cf6,0x734e,0x6e38,0x6279,0x7565,0x77ed,0x5e7b,0x6c99);
U8(0x6563,0x6575,0x9109,0x518a,0x8f2a,0x671d,0x7a97,0x5fcd);
U8(0x6cb3,0x85cf,0x885d,0x6df7,0x552f,0x4e7e,0x51a0,0x719f);
U8(0x9df9,0x86cb,0x5c0a,0x68c4,0x656c,0x5b63,0x5a5a,0x7e23);
U8(0x7dca,0x4f2f,0x7533,0x8863,0x8cfc,0x50c5,0x5e33,0x5c64);
U8(0x79cb,0x731c,0x504f,0x93e1,0x98df,0x559d,0x5077,0x8d0a);
U8(0x72af,0x52c7,0x9846,0x59d3,0x675f,0x6de1,0x8a69,0x5609);
U8(0x66c9,0x501f,0x5f92,0x6d32,0x64c1,0x5e8f,0x6176,0x7e3e);
U8(0x795d,0x7345,0x570d,0x9918,0x79c1,0x9b25,0x67d4,0x6f02);
U8(0x5bcc,0x79c0,0x7bc4,0x907f,0x8f1d,0x8b6f,0x5b64,0x7b28);
U8(0x62ec,0x5438,0x7aef,0x79fb,0x5ef3,0x84cb,0x672b,0x5bdf);
U8(0x6297,0x63ee,0x4e56,0x7a4d,0x63d2,0x9a5a,0x8521,0x5fe0);
U8(0x6108,0x96ea,0x5de8,0x78a9,0x745e,0x51e1,0x6731,0x7c43);
U8(0x4e1f,0x76e4,0x5076,0x6e2f,0x5bae,0x5e25,0x5b8f,0x96c5);
U8(0x8ca8,0x9192,0x865b,0x907a,0x639b,0x900f,0x7206,0x70c8);
U8(0x6ec5,0x6750,0x62cd,0x4f11,0x8a8c,0x5713,0x9280,0x4e92);
U8(0x98ef,0x638c,0x8c6a,0x66ff,0x6311,0x9802,0x718a,0x7d2f);
U8(0x5740,0x5065,0x4fe0,0x96de,0x56f0,0x8afe,0x8f14,0x4f34);
U8(0x7389,0x6d17,0x654f,0x81e8,0x8857,0x964d,0x5538,0x6190);
U8(0x8607,0x7c73,0x6e1b,0x64cd,0x760b,0x8f9b,0x5b87,0x805a);
U8(0x516e,0x5f7c,0x63a1,0x5c46,0x5c3c,0x8f29,0x85e5,0x9a19);
U8(0x7da0,0x7ae5,0x7e2e,0x7b26,0x7372,0x9ed8,0x90ce,0x78b0);
U8(0x7981,0x5a46,0x54a7,0x80a1,0x62bd,0x780d,0x6068,0x80af);
U8(0x520a,0x6ce1,0x6f38,0x92fc,0x5ee2,0x6dda,0x6a1e,0x8d0f);
U8(0x6b50,0x8ca1,0x8a02,0x9014,0x89f8,0x8ce2,0x5091,0x812b);
U8(0x7bb1,0x4f48,0x4ed9,0x51b0,0x6790,0x84bc,0x53ad,0x7c4d);
U8(0x5805,0x54ed,0x590f,0x61f6,0x6a4b,0x7a69,0x9732,0x8a62);
U8(0x68ee,0x7d72,0x5875,0x6163,0x6fe4,0x8abc,0x4f54,0x8336);
U8(0x8cf4,0x968e,0x8c50,0x8c9d,0x6691,0x90f5,0x5371,0x5bd2);
U8(0x68b0,0x96b1,0x7238,0x6c7d,0x6158,0x65c5,0x6751,0x4e88);
U8(0x632f,0x4ea1,0x74f6,0x906d,0x54f2,0x96f6,0x7d0d,0x73cd);
U8(0x4e43,0x5e2d,0x54e9,0x912d,0x5439,0x9aa8,0x7159,0x80e1);
U8(0x502b,0x6d2a,0x6416,0x5a18,0x82b3,0x7d22,0x5cf0,0x7e54);
U8(0x60e0,0x676f,0x6d6e,0x72fc,0x9694,0x6930,0x79df,0x5c3e);
U8(0x5ffd,0x8907,0x838a,0x6b3e,0x83dc,0x526f,0x4f01,0x6298);
U8(0x64fe,0x63da,0x9f20,0x5192,0x7661,0x9f13,0x523a,0x983b);
U8(0x55b5,0x91dd,0x9738,0x66ab,0x4f0a,0x6c89,0x5634,0x5eab);
U8(0x6094,0x9670,0x5bc2,0x501a,0x63a2,0x7956,0x6469,0x8ddd);
```

```
U8(0x5269,0x904d,0x87f2,0x71d2,0x9801,0x9686,0x5f31,0x8c46);
U8(0x5ef6,0x856d,0x63e1,0x5be7,0x662d,0x76df,0x8986,0x61b2);
U8(0x79d8,0x8010,0x6101,0x4e01,0x64a5,0x6c88,0x6607,0x9aee);
U8(0x54c0,0x9f9c,0x7709,0x56fa,0x5377,0x984f,0x9592,0x7a0d);
U8(0x6b98,0x642d,0x4eac,0x50bb,0x8033,0x6620,0x8a17,0x6069);
U8(0x9d3b,0x5999,0x8faf,0x5a92,0x5435,0x64ec,0x9055,0x6182);
U8(0x684c,0x62fc,0x92b7,0x85c9,0x80d6,0x5c4b,0x57df,0x8239);
U8(0x52de,0x6d1e,0x5be2,0x4e95,0x5f90,0x5ddd,0x5948,0x6236);
U8(0x4e58,0x984d,0x500d,0x8fb2,0x64fa,0x585e,0x6a6b,0x8cc0);
U8(0x7687,0x62d6,0x6028,0x7259,0x9f4a,0x9003,0x969c,0x8cfa);
U8(0x6efe,0x5e72,0x8b5c,0x70cf,0x77ad,0x6234,0x642c,0x9b06);
U8(0x74dc,0x8feb,0x5687,0x5976,0x68cb,0x9b27,0x76db,0x6478);
U8(0x5c3a,0x4f19,0x67d3,0x4f69,0x5e8a,0x9177,0x5b6b,0x7260);
U8(0x5c41,0x9189,0x9707,0x9396,0x51f1,0x640d,0x6f6e,0x5f79);
U8(0x6cf0,0x4f0d,0x51ac,0x6843,0x8070,0x4e59,0x8ff4,0x7af6);
U8(0x5d07,0x649e,0x53b2,0x6de8,0x727d,0x7f8a,0x60d1,0x6eaa);
U8(0x5e3d,0x7720,0x5df7,0x6e9d,0x4fd7,0x5b8b,0x98c4,0x6dfa);
U8(0x737b,0x9eb5,0x8266,0x6ed1,0x699c,0x5708,0x5ead,0x7e31);
U8(0x9583,0x84ee,0x9ea5,0x90a6,0x8000,0x65cb,0x95b1,0x8ad2);
U8(0x7d2b,0x651d,0x5100,0x59a8,0x5be9,0x63cf,0x6d3d,0x6bc0);
U8(0x9010,0x966a,0x586b,0x8e0f,0x8dcc,0x85e4,0x6155,0x52d2);
U8(0x95c6,0x6b8a,0x59ca,0x6191,0x5bde,0x5cb8,0x82ac,0x93ae);
U8(0x6c60,0x62d2,0x845b,0x5dde,0x92d2,0x5bec,0x62f3,0x4e8e);
U8(0x51fd,0x9a45,0x540e,0x88c1,0x87f9,0x963b,0x90aa,0x9769);
U8(0x9817,0x81e5,0x660c,0x5b5f,0x822a,0x9059,0x7fd4,0x880d);
U8(0x606d,0x5967,0x9f4b,0x775b,0x69cd,0x5854,0x9589,0x9813);
U8(0x7cca,0x4ef0,0x65d7,0x52f5,0x539a,0x9b42,0x51c6,0x4e38);
U8(0x5b99,0x9038,0x88d5,0x8212,0x8a2a,0x517c,0x85a9,0x98f2);
U8(0x8d64,0x95a3,0x5eb8,0x556a,0x6436,0x52ff,0x7802,0x7dd2);
U8(0x4e0c,0x7a79,0x5118,0x6383,0x6148,0x6b04,0x5617,0x52f8);
U8(0x9bae,0x596e,0x8cdc,0x83e9,0x558a,0x95e1,0x5e7d,0x64f4);
U8(0x6c61,0x5f70,0x6beb,0x7965,0x76dc,0x8a3c,0x7f70,0x756a);
U8(0x5c16,0x7070,0x72d0,0x8861,0x64ce,0x6c6a,0x76e3,0x7c4c);
U8(0x9072,0x8a87,0x90b1,0x7a05,0x6d69,0x62c6,0x7d1b,0x543e);
U8(0x6258,0x59d1,0x54b1,0x6f0f,0x8a95,0x6dd1,0x9b31,0x7832);
U8(0x67cf,0x7246,0x7aa9,0x50be,0x8fad,0x79d2,0x7fbd,0x6e21);
U8(0x6c96,0x5ed6,0x8f1b,0x9f61,0x57f9,0x9080,0x7518,0x888b);
U8(0x675c,0x9178,0x80f8,0x908f,0x8c48,0x6dbc,0x80a5,0x8c37);
U8(0x7378,0x5a03,0x5b54,0x9677,0x4f30,0x59bb,0x5f91,0x73ab);
U8(0x6f5b,0x6676,0x80de,0x715e,0x8106,0x62b9,0x7336,0x7bc9);
U8(0x5442,0x788e,0x5674,0x622a,0x65e6,0x5a66,0x5783,0x8352);
U8(0x5e63,0x573e,0x4e32,0x58ef,0x4f38,0x50b2,0x5f26,0x6ce5);
U8(0x7e41,0x9b6f,0x5fcc,0x5104,0x53ed,0x5766,0x6065,0x871c);
U8(0x6b3a,0x6368,0x6021,0x73b2,0x81ed,0x5510,0x6e6f,0x7aae);
U8(0x6089,0x52c1,0x62d4,0x751c,0x8499,0x901b,0x6735,0x6458);
U8(0x81bd,0x7cdf,0x817f,0x5587,0x919c,0x8e8d,0x7e73,0x6cc9);
U8(0x6fa4,0x78a7,0x978b,0x97d3,0x5f65,0x6668,0x8389,0x99d5);
U8(0x731b,0x7de9,0x5761,0x70b8,0x6012,0x53ec,0x6355,0x5e7c);
U8(0x827e,0x6046,0x5272,0x626f,0x76fe,0x4e48,0x5410,0x660f);
U8(0x8fa8,0x8ed2,0x5deb,0x4e39,0x9451,0x5f4e,0x8ce6,0x576a);
U8(0x4fb5,0x80ce,0x60a0,0x85aa,0x522a,0x5f6c,0x609f,0x73e0);
U8(0x52e4,0x7470,0x690d,0x75f4,0x7919,0x655d,0x76c3,0x76fc);
U8(0x4ea8,0x901d,0x526a,0x78e8,0x903c,0x6fc3,0x6d1b,0x4e4f);
U8(0x64cb,0x59c6,0x75c7,0x5acc,0x8eb2,0x58c1,0x91e3,0x6696);
U8(0x61be,0x905c,0x6109,0x66f0,0x91c7,0x6377,0x5606,0x7384);
U8(0x6ef4,0x6263,0x62b5,0x5954,0x7fc1,0x8e5f,0x83f2,0x722c);
```

```
        U8(0x68da,0x5915,0x60f1,0x6212,0x7272,0x6cf3,0x6070,0x6ec4);
        U8(0x6851,0x8523,0x6f54,0x7126,0x52c9,0x596a,0x840a,0x5496);
        U8(0x8caa,0x8e22,0x6284,0x8de1,0x937e,0x6daf,0x7bad,0x8a93);
        U8(0x7a4c,0x6d89,0x7a3f,0x7f72,0x7642,0x790e,0x50d1,0x6db5);
        U8(0x9727,0x7c89,0x4ff1,0x5fb9,0x8b00,0x5c60,0x72a7,0x5537);
        U8(0x7e5e,0x58fd,0x723a,0x9a37,0x6c0f,0x7ffc,0x4e91,0x809a);
        U8(0x541d,0x6881,0x54aa,0x7e6a,0x5ec9,0x98fd,0x9cf4,0x8c8c);
        U8(0x9739,0x70ae,0x6350,0x5561,0x708e,0x55e8,0x8ca2,0x66fc);
        U8(0x58e2,0x9f3b,0x5949,0x9234,0x8017,0x6062,0x878d,0x7b4b);
        U8(0x64bf,0x5112,0x98fe,0x798d,0x4faf,0x552c,0x6316,0x5e45);
        U8(0x77a7,0x5ef7,0x54a6,0x6490,0x67f3,0x8584,0x9ece,0x6372);
        U8(0x5d50,0x540a,0x9742,0x64e0,0x89bd,0x8fea,0x51cc,0x832b);
        U8(0x71c3,0x746a,0x5f18,0x77db,0x6566,0x745c,0x9326,0x6953);
        U8(0x8cd3,0x68af,0x742a,0x7344,0x5074,0x54ac,0x58a8,0x60b6);
        U8(0x6674,0x6016,0x55ac,0x9f52,0x5dba,0x4ef2,0x97fb,0x65ed);
        U8(0x7d9c,0x5154,0x8e64,0x59ff,0x9081,0x50a2,0x8b7d,0x8932);
        U8(0x5b5d,0x4fc3,0x99db,0x4ec7,0x7c97,0x59ae,0x7267,0x821f);
        U8(0x7cd5,0x6bbc,0x8776,0x5147,0x934b,0x61fc,0x92b3,0x541f);
        U8(0x8667,0x679d,0x800d,0x95ca,0x75bc,0x62fe,0x6bc5,0x5237);
        U8(0x5857,0x593e,0x88c2,0x9640,0x54c9,0x9130,0x6085,0x5951);
        U8(0x8f5f,0x7235,0x5c4f,0x99d0,0x8ced,0x8702,0x6247,0x7f9e);
        U8(0x74e6,0x758f,0x6b3d,0x6597,0x5764,0x582a,0x6d29,0x9006);
        U8(0x62ac,0x54ce,0x9905,0x5f4c,0x8cab,0x7db1,0x60f9,0x7ff9);
        U8(0x58ae,0x72e0,0x9811,0x838e,0x64e6,0x8d08,0x51dd,0x6ecb);
        U8(0x9756,0x5582,0x5141,0x5f6d,0x633a,0x62cb,0x611a,0x5f81);
        U8(0x8fb1,0x6717,0x4f59,0x6170,0x79aa,0x86d9,0x79e6,0x707d);
        U8(0x5075,0x8b9a,0x6905,0x5925,0x7897,0x4e19,0x5352,0x8377);
        U8(0x62ab,0x8da8,0x52ab,0x8fc5,0x7b1b,0x70e4,0x80a9,0x77e3);
        U8(0x9077,0x6c5d,0x9e7f,0x7f38,0x617e,0x7210,0x9b45,0x4e08);
        U8(0x5de1,0x63aa,0x79e4,0x6cbf,0x75d5,0x6feb,0x6f32,0x8170);
        U8(0x8ce4,0x59a5,0x6db2,0x9d5d,0x8ecc,0x53d4,0x9675,0x8c9e);
        U8(0x626d,0x6f20,0x7955,0x8766,0x6349,0x8b20,0x6c1b,0x81a0);
        U8(0x905e,0x9716,0x96ef,0x9ad2,0x5006,0x65fa,0x6749,0x71d5);
        U8(0x5bf8,0x8302,0x5faa,0x5c4d,0x947d,0x6643,0x6084,0x7fc5);
        U8(0x8292,0x659c,0x5ec1,0x5466,0x6e9c,0x9db4,0x55aa,0x76d2);
        U8(0x6454,0x6127,0x7051,0x6d25,0x76c8,0x8108,0x82d7,0x5821);
        U8(0x5211,0x53c9,0x6492,0x866b,0x7e8f,0x9075,0x81e3,0x6afb);

// SC : most frequent 2048 SC minus TC ( 768 code points )

        U8(0x8cc7,0x9019,0x500b,0x6703,0x70ba,0x4f86,0x5b78,0x6642);
        U8(0x8aaa,0x6c92,0x554f,0x904e,0x8acb,0x5011,0x9ebc,0x9084);
        U8(0x96fb,0x5c0d,0x6a5f,0x8a0a,0x570b,0x767c,0x7121,0x7576);
        U8(0x55ce,0x73fe,0x9ede,0x984c,0x6a23,0x7d93,0x8b1d,0x958b);
        U8(0x8a71,0x8207,0x5be6,0x611b,0x83ef,0x52d5,0x61c9,0x7a2e);
        U8(0x8eca,0x9577,0x95dc,0x9593,0x9cf3,0x89ba,0x8a72,0x9032);
        U8(0x5c07,0x5169,0x8ad6,0x9f8d,0x5225,0x9ad4,0x7d66,0x807d);
        U8(0x898b,0x88e1,0x6771,0x98a8,0x7063,0x539f,0x7db2,0x5340);
        U8(0x932f,0x6a02,0x66f8,0x8b93,0x9078,0x8f03,0x6578,0x5167);
        U8(0x5834,0x5f9e,0x6b61,0x8a8d,0x5e7e,0x96e3,0x982d,0x8a31);
        U8(0x8cb7,0x7d71,0x8a18,0x865f,0x8655,0x8a08,0x5e2b,0x4e26);
        U8(0x5f35,0x8ab0,0x7d50,0x9023,0x8a2d,0x9673,0x5831,0x8f49);
        U8(0x9ee8,0x8b8a,0x6c23,0x8a66,0x55ae,0x7fa9,0x6230,0x968a);
        U8(0x537b,0x696d,0x5beb,0x8072,0x81fa,0x54e1,0x8a0e,0x5247);
        U8(0x6a94,0x59b3,0x8208,0x5f37,0x8b70,0x8fa6,0x50f9,0x7e3d);
        U8(0x50b3,0x6c7a,0x7d44,0x842c,0x7d1a,0x6a19,0x7368,0x8c93);
```

```
U8(0x9580,0x8a9e,0x7dda,0x99ac,0x9ec3,0x53c3,0x8996,0x89c0);
U8(0x806f,0x8166,0x5152,0x9322,0x9304,0x5c08,0x63db,0x9060);
U8(0x5099,0x5e6b,0x78ba,0x88dd,0x985e,0x8b1b,0x756b,0x8a34);
U8(0x5e36,0x5716,0x8cbb,0x8cfd,0x8b58,0x908a,0x96d6,0x904b);
U8(0x52d9,0x6b0a,0x9a57,0x8b80,0x98db,0x6eff,0x5922,0x8ab2);
U8(0x8ecd,0x7d04,0x9858,0x9054,0x5920,0x8ac7,0x8853,0x8ce3);
U8(0x52dd,0x6e96,0x6cc1,0x96e2,0x5c0e,0x5718,0x8abf,0x8b49);
U8(0x50b7,0x525b,0x6bba,0x8edf,0x689d,0x7d55,0x7a31,0x64da);
U8(0x76e1,0x805e,0x5712,0x7522,0x96d9,0x7d05,0x57f7,0x96f2);
U8(0x904a,0x9808,0x8cea,0x8077,0x5fa9,0x8f38,0x7bc0,0x898f);
U8(0x7562,0x71b1,0x9928,0x614b,0x89aa,0x4e82,0x5bf6,0x64ca);
U8(0x8209,0x7d42,0x65b7,0x8f15,0x74b0,0x7c21,0x97ff,0x96a8);
U8(0x7df4,0x7e8c,0x9b5a,0x6975,0x967d,0x5e79,0x7fd2,0x7f85);
U8(0x62c9,0x6a13,0x5ee3,0x7169,0x9ce5,0x986f,0x78bc,0x8056);
U8(0x58de,0x9810,0x773e,0x8cac,0x722d,0x8aa4,0x9806,0x8cb4);
U8(0x8ca0,0x58d3,0x9069,0x4fc2,0x6e2c,0x61f7,0x8fce,0x55da);
U8(0x8a55,0x7d30,0x91ab,0x5c6c,0x6200,0x6557,0x694a,0x9748);
U8(0x969b,0x9a0e,0x9805,0x6232,0x71df,0x6b77,0x723e,0x7de8);
U8(0x8b77,0x88dc,0x64c7,0x6b72,0x96bb,0x9818,0x5c0b,0x6eab);
U8(0x990a,0x8b02,0x7570,0x512a,0x721b,0x60e1,0x72c0,0x7dad);
U8(0x9678,0x5433,0x8a5e,0x5287,0x975c,0x4e9e,0x6c5f,0x9663);
U8(0x56b4,0x58d8,0x52e2,0x862d,0x7de3,0x9031,0x5ee0,0x7c3d);
U8(0x7ffb,0x883b,0x7d61,0x8f2f,0x5275,0x8d99,0x820a,0x96dc);
U8(0x9e97,0x79ae,0x8173,0x528d,0x616e,0x7d39,0x5289,0x61b6);
U8(0x8af8,0x7f75,0x7d14,0x85dd,0x8aa0,0x7e7c,0x7d00,0x91cb);
U8(0x88fd,0x9375,0x8d95,0x7b46,0x8a3b,0x6a39,0x9435,0x69ae);
U8(0x6b78,0x8449,0x5f48,0x885b,0x9298,0x584a,0x6f22,0x8cde);
U8(0x8f09,0x96aa,0x9418,0x69cb,0x56c9,0x87a2,0x5049,0x85a6);
U8(0x555f,0x71c8,0x6aa2,0x5abd,0x8c6c,0x6fdf,0x8a13,0x85cd);
U8(0x5283,0x64d4,0x7d19,0x8cbc,0x7f77,0x8a73,0x5354,0x9867);
U8(0x81c9,0x5cf6,0x734e,0x6575,0x9109,0x518a,0x8f2a,0x885d);
U8(0x9df9,0x68c4,0x7e23,0x7dca,0x8cfc,0x50c5,0x5e33,0x5c64);
U8(0x93e1,0x8d0a,0x9846,0x6de1,0x8a69,0x66c9,0x64c1,0x6176);
U8(0x7e3e,0x7345,0x570d,0x9918,0x9b25,0x7bc4,0x8f1d,0x8b6f);
U8(0x5ef3,0x84cb,0x63ee,0x7a4d,0x9a5a,0x78a9,0x7c43,0x4e1f);
U8(0x76e4,0x5bae,0x5e25,0x8ca8,0x865b,0x907a,0x639b,0x6ec5);
U8(0x8a8c,0x5713,0x9280,0x98ef,0x9802,0x4fe0,0x96de,0x8afe);
U8(0x8f14,0x81e8,0x5538,0x6190,0x8607,0x6e1b,0x760b,0x63a1);
U8(0x5c46,0x8f29,0x85e5,0x9a19,0x7da0,0x7e2e,0x7372,0x6f38);
U8(0x92fc,0x5ee2,0x6dda,0x6a1e,0x8d0f,0x6b50,0x8ca1,0x8a02);
U8(0x89f8,0x8ce2,0x5091,0x812b,0x4f48,0x84bc,0x53ad,0x5805);
U8(0x61f6,0x6a4b,0x7a69,0x8a62,0x68ee,0x7d72,0x5875,0x6163);
U8(0x6fe4,0x8abc,0x4f54,0x8cf4,0x968e,0x8c50,0x8c9d,0x90f5);
U8(0x96b1,0x6158,0x7d0d,0x912d,0x7159,0x502b,0x6416,0x7e54);
U8(0x676f,0x8907,0x838a,0x64fe,0x63da,0x7661,0x983b,0x55b5);
U8(0x91dd,0x66ab,0x5eab,0x9670,0x87f2,0x71d2,0x9801,0x856d);
U8(0x5be7,0x61b2,0x64a5,0x6607,0x9aee,0x9f9c,0x984f,0x9592);
U8(0x6b98,0x8a17,0x9d3b,0x8faf,0x64ec,0x9055,0x6182,0x92b7);
U8(0x52de,0x5be2,0x6236,0x984d,0x8fb2,0x64fa,0x6a6b,0x8cc0);
U8(0x9f4a,0x8cfa,0x6efe,0x8b5c,0x70cf,0x77ad,0x9b06,0x5687);
U8(0x9b27,0x5b6b,0x7260,0x9396,0x51f1,0x640d,0x8070,0x8ff4);
U8(0x7af6,0x53b2,0x6de8,0x727d,0x6e9d,0x98c4,0x6dfa,0x737b);
U8(0x9eb5,0x8266,0x7e31,0x9583,0x84ee,0x9ea5,0x95b1,0x8ad2);
U8(0x651d,0x5100,0x5be9,0x6bc0,0x95c6,0x6191,0x93ae,0x92d2);
U8(0x5bec,0x62f3,0x9a45,0x9817,0x81e5,0x9059,0x880d,0x5967);
U8(0x9f4b,0x69cd,0x9589,0x9813,0x52f5,0x8a2a,0x85a9,0x98f2);
```

```
        U8(0x95a3,0x6436,0x7dd2,0x4e0c,0x5118,0x6383,0x6b04,0x5617);
        U8(0x52f8,0x9bae,0x596e,0x8cdc,0x95e1,0x64f4,0x76dc,0x8a3c);
        U8(0x7f70,0x76e3,0x7c4c,0x9072,0x8a87,0x7a05,0x7d1b,0x8a95);
        U8(0x9b31,0x7832,0x7246,0x7aa9,0x50be,0x8fad,0x6c96,0x8f1b);
        U8(0x9f61,0x908f,0x8c48,0x6dbc,0x7378,0x5f91,0x6f5b,0x7336);
        U8(0x7bc9,0x5442,0x5674,0x5a66,0x5e63,0x58ef,0x9b6f,0x5104);
        U8(0x6065,0x6368,0x6e6f,0x7aae,0x52c1,0x81bd,0x919c,0x8e8d);
        U8(0x7e73,0x6fa4,0x97d3,0x5f65,0x99d5,0x7de9,0x6012,0x6046);
        U8(0x8ed2,0x9451,0x5f4e,0x8ce6,0x522a,0x7919,0x76c3,0x6fc3);
        U8(0x64cb,0x91e3,0x905c,0x5606,0x8e5f,0x60f1,0x6ec4,0x8523);
        U8(0x6f54,0x596a,0x840a,0x8caa,0x6284,0x8de1,0x937e,0x7a4c);
        U8(0x7642,0x790e,0x50d1,0x6db5,0x9727,0x5fb9,0x8b00,0x72a7);
        U8(0x7e5e,0x58fd,0x723a,0x9a37,0x7e6a,0x98fd,0x9cf4,0x8ca2);
        U8(0x58e2,0x9234,0x64bf,0x98fe,0x798d,0x6490,0x6372,0x5d50);
        U8(0x9742,0x64e0,0x89bd,0x746a,0x9326,0x6953,0x8cd3,0x7344);
        U8(0x5074,0x60b6,0x55ac,0x9f52,0x5dba,0x97fb,0x7d9c,0x8e64);
        U8(0x9081,0x50a2,0x8b7d,0x8932,0x99db,0x6bbc,0x5147,0x934b);
        U8(0x61fc,0x92b3,0x8667,0x95ca,0x5857,0x593e,0x9130,0x6085);
        U8(0x8f5f,0x99d0,0x8ced,0x6b3d,0x6d29,0x9905,0x5f4c,0x8cab);
        U8(0x7db1,0x7ff9,0x58ae,0x9811,0x8d08,0x62cb,0x79aa,0x707d);
        U8(0x5075,0x8b9a,0x8da8,0x9077,0x617e,0x7210,0x6feb,0x6f32);
        U8(0x8ce4,0x9d5d,0x8ecc,0x8c9e,0x7955,0x8766,0x8b20,0x81a0);
        U8(0x905e,0x9ad2,0x5006,0x5faa,0x5c4d,0x947d,0x5ec1,0x9db4);
        U8(0x55aa,0x7051,0x8108,0x7e8f,0x6afb,0x6e3e,0x5641,0x5132);

        return s;
}

u_code_point restore_order_unihan(u_code_point z) {
        return reorder_unihan(z);
}


#define MAPCHAR(x,A,B,bytes) if(A<=x && x< (A+bytes)) \
        return(x+(B-A)); if(B<=x && x< (B+bytes)) return(x+(A-B))
#define MAP16BL(x,A,B,block) if(A<=x && x< (A+(block<<4))) \
        return(x+(B-A)); if(B<=x && x< (B+(block<<4))) \
        return(x+(A-B))

u_code_point reorder_latins(u_code_point s) {
        MAP16BL(s,0x0100,0x0000,3); // Latin Extension A
        MAP16BL(s,0x0130,0x0080,2);
        MAP16BL(s,0x0150,0x00A0,1);
        MAP16BL(s,0x0300,0x00B0,3); // Combining Diacritical Marks
        MAPCHAR(s,0x0070,0x0060,1); // p,`
        MAPCHAR(s,0x0072,0x006A,1); // r,j
        MAPCHAR(s,0x0073,0x006B,1); // s,k
        MAPCHAR(s,0x0074,0x0066,1); // t,f
        MAPCHAR(s,0x0075,0x0067,1); // u,g
        MAPCHAR(s,0x0050,0x0040,1); // P,@  UPPER
        MAPCHAR(s,0x0052,0x004A,1); // R,J  UPPER
        MAPCHAR(s,0x0053,0x004B,1); // S,K  UPPER
        MAPCHAR(s,0x0054,0x0046,1); // T,F  UPPER
        MAPCHAR(s,0x0055,0x0047,1); // U,G  UPPER
        MAPCHAR(s,0x0160,0x003A,6); // Latin Extension A
        MAPCHAR(s,0x0166,0x005B,5);
```

```
        MAPCHAR(s,0x016B,0x007B,5);
        return s;
}

u_code_point restore_order_latins(u_code_point z) {
        return reorder_latins(z);
}

u_code_point reorder(u_code_point s) {
        if(isHANGUL(s)) return reorder_hangul(s);
        if(isUNIHAN(s)) return reorder_unihan(s);
        if(isLatins(s)) return reorder_latins(s);
        return s;
}
u_code_point restore_order(u_code_point s) {
        if(isHANGUL(s)) return restore_order_hangul(s);
        if(isUNIHAN(s)) return restore_order_unihan(s);
        if(isLatins(s)) return restore_order_latins(s);
        return s;
}


/* Encoder: */

enum dude_status dude_encode(
  unsigned int input_length,
  const u_code_point input[],
  const unsigned char uppercase_flags[],
  unsigned int *output_size,
  char output[] )
{
  unsigned int max_out, in, out, k, j;
  u_code_point prev, codept, diff, tmp;
  char shift;

  prev = 0x60;
  max_out = *output_size;

  for (in = out = 0;  in < input_length;  ++in) {

    /* At the start of each iteration, in and out are the number of */
    /* items already input/output, or equivalently, the indices of  */
    /* the next items to be input/output.                           */

    codept = input[in];

    if (codept == 0x2D) {
      /* Hyphen-minus stands for itself. */
      if (max_out - out < 1) return dude_big_output;
      output[out++] = 0x2D;
      continue;
    }

    codept = reorder(codept); // by LSB

    diff = prev^codept;
```

```
    /* Compute the number of base-32 characters (k): */
    for (tmp = diff >> 4, k = 1;  tmp != 0;  ++k, tmp >>= 4);
    //fprintf(stderr,"diff %x,%x = prev %x ^ codept %x \n",
    //        k,diff,prev,codept);

    if (max_out - out < k) return dude_big_output;
    shift = uppercase_flags && uppercase_flags[in] ? 32 : 0;
    /* shift controls the case of the last base-32 digit. */

    /* Each quintet has the form 1xxxx except the last is 0xxxx. */
    /* Computing the base-32 digits in reverse order is easiest. */

    out += k;
    output[out - 1] = base32[diff & 0xF] - shift;

    for (j = 2;  j <= k;  ++j) {
      diff >>= 4;
      output[out - j] = base32[0x10 | (diff & 0xF)];
    }

    prev = codept;
  }

  /* Append the null terminator: */
  if (max_out - out < 1) return dude_big_output;
  output[out++] = 0;

  *output_size = out;
  return dude_success;
}


/* Decoder: */

enum dude_status dude_decode(
  enum case_sensitivity case_sensitivity,
  char scratch_space[],
  const char input[],
  unsigned int *output_length,
  u_code_point output[],
  unsigned char uppercase_flags[] )
{
  u_code_point prev, q, diff;
  char c;
  unsigned int max_out, in, out, scratch_size;
  enum dude_status status;

  prev = 0x60;
  max_out = *output_length;

  for (c = input[in = 0], out = 0;  c != 0;  c = input[++in], ++out) {

    /* At the start of each iteration, in and out are the number of */
    /* items already input/output, or equivalently, the indices of  */
```

```
    /* the next items to be input/output.                           */

    if (max_out - out < 1) return dude_big_output;

    if (c == 0x2D) output[out] = c;  /* hyphen-minus is literal */
    else {
      /* Base-32 sequence.  Decode quintets until 0xxxx is found: */

      for (diff = 0;  ;  c = input[++in]) {
        q = base32_decode(c);
        if (q == base32_invalid){ return dude_bad_input; };
        diff = (diff << 4) | (q & 0xF);
        if (q >> 4 == 0) break;
      }

      // prev = output[out] = prev ^ diff;
      prev = prev ^ diff;
      output[out] = restore_order(prev); // LSB

    }

    /* Case of last character determines uppercase flag: */
    if (uppercase_flags) uppercase_flags[out] = c >= 65 && c <= 90;
  }

  /* Enforce the uniqueness of the encoding by re-encoding */
  /* the output and comparing the result to the input:     */

  scratch_size = ++in;
  status = dude_encode(out, output, uppercase_flags,
                       &scratch_size, scratch_space);
  if (status != dude_success || scratch_size != in ||
      unequal(case_sensitivity, scratch_space, input)
     ) return dude_bad_input;
  *output_length = out;
  return dude_success;
}


/*****************************************************************/
/* Wrapper for testing (would normally go in a separate .c file): */

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* For testing, we'll just set some compile-time limits rather than */
/* use malloc(), and set a compile-time option rather than using a  */
/* command-line option.                                             */

enum {
  unicode_max_length = 256,
  ace_max_size = 256,
  test_case_sensitivity = case_insensitive
                          /* suitable for host names */
```

```
};


static void usage(char **argv)
{
  fprintf(stderr,
    "%s -e reads code points and writes a DUDE string.\n"
    "%s -d reads a DUDE string and writes code points.\n"
    "Input and output are plain text in the native character set.\n"
    "Code points are in the form u+hex separated by whitespace.\n"
    "A DUDE string is a newline-terminated sequence of LDH
characters\n"
    "(without any signature).\n"
    "The case of the u in u+hex is the force-to-uppercase flag.\n"
    , argv[0], argv[0]);
  exit(EXIT_FAILURE);
}


static void fail(const char *msg)
{
  fputs(msg,stderr);
  exit(EXIT_FAILURE);
}

static const char too_big[] =
  "input or output is too large, recompile with larger limits\n";
static const char invalid_input[] = "invalid input\n";
static const char io_error[] = "I/O error\n";


/* The following string is used to convert LDH      */
/* characters between ASCII and the native charset: */

static const char ldh_ascii[] =
  "................"
  "................"
  ".............-.."
  "0123456789......"
  ".ABCDEFGHIJKLMNO"
  "PQRSTUVWXYZ....."
  ".abcdefghijklmno"
  "pqrstuvwxyz";


int main(int argc, char **argv)
{
  enum dude_status status;
  int r;
  char *p;

  if (argc != 2) usage(argv);
  if (argv[1][0] != '-') usage(argv);
  if (argv[1][2] != 0) usage(argv);

  if (argv[1][1] == 'e') {
```

```
  u_code_point input[unicode_max_length];
  unsigned long codept;
  unsigned char uppercase_flags[unicode_max_length];
  char output[ace_max_size], uplus[3];
  unsigned int input_length, output_size, i;

  /* Read the input code points: */

  input_length = 0;

  for (;;) {
    r = scanf("%2s%lx", uplus, &codept);
    if (ferror(stdin)) fail(io_error);
    if (r == EOF || r == 0) break;

    if (r != 2 || uplus[1] != '+' || codept > (u_code_point)-1) {
      fail(invalid_input);
    }

    if (input_length == unicode_max_length) fail(too_big);

    if (uplus[0] == 'u') uppercase_flags[input_length] = 0;
    else if (uplus[0] == 'U') uppercase_flags[input_length] = 1;
    else fail(invalid_input);

    input[input_length++] = codept;
  }

  /* Encode: */

  output_size = ace_max_size;
  status = dude_encode(input_length, input, uppercase_flags,
                       &output_size, output);
  if (status == dude_bad_input) fail(invalid_input);
  if (status == dude_big_output) fail(too_big);
  assert(status == dude_success);

  /* Convert to native charset and output: */

  for (p = output;  *p != 0;  ++p) {
    i = *p;
    assert(i <= 122 && ldh_ascii[i] != '.');
    *p = ldh_ascii[i];
  }

  r = puts(output);
  if (r == EOF) fail(io_error);
  return EXIT_SUCCESS;
}

if (argv[1][1] == 'd') {
  char input[ace_max_size], scratch[ace_max_size], *pp;
  u_code_point output[unicode_max_length];
  unsigned char uppercase_flags[unicode_max_length];
  unsigned int input_length, output_length, i;
```

```
    /* Read the DUDE input string and convert to ASCII: */

    fgets(input, ace_max_size, stdin);
    if (ferror(stdin)) fail(io_error);
    if (feof(stdin)) fail(invalid_input);
    input_length = strlen(input);
    if (input[input_length - 1] != '\n') fail(too_big);
    input[--input_length] = 0;

    for (p = input;  *p != 0;  ++p) {
      pp = strchr(ldh_ascii, *p);
      if (pp == 0) fail(invalid_input);
      *p = pp - ldh_ascii;
    }

    /* Decode: */

    output_length = unicode_max_length;
    status = dude_decode(test_case_sensitivity, scratch, input,
                         &output_length, output, uppercase_flags);
    if (status == dude_bad_input) fail(invalid_input);
    if (status == dude_big_output) fail(too_big);
    assert(status == dude_success);

    /* Output the result: */

    for (i = 0;  i < output_length;  ++i) {
      r = printf("%s+%04lX\n",
                 uppercase_flags[i] ? "U" : "u",
                 (unsigned long) output[i] );
      if (r < 0) fail(io_error);
    }

    return EXIT_SUCCESS;
  }

  usage(argv);
  return EXIT_SUCCESS;  /* not reached, but quiets compiler warning */
}
```